INSTITUT D'INTELLIGENCE ARTIFICIELLE
INFORMATIQUE MUSICALE
UNIVERSITÉ PARIS 8

FEVRIER 1973

LE MANUEL LISP 510

PATRICK GREUSSAY

UNIVERSITÉ PARIS 8 / RECHERCHE
GROUPE 2 . ÉQUIPE 9 .

- GENERALITÉS -

L'interprète tourne en mode EVAL : il lit un atome ou un appel de fonction, l'évalue, imprime le résultat de l'évaluation, et recommence. Quelque chose comme ça :

```
(WHILE T
        (PRINT (EVAL (READ))))
```

On peut entrer des S-expressions sur cartes ou au clavier de la machine à écrire (MAE). Sur cartes, on peut écrire en format libre sur toute la carte. Au clavier, la MAE indique qu'elle attend une ligne en tapant " ? ". On tape alors à la suite.

FORME DES ATOMES :

- Ça peut être un nombre, i.e. une suite de chiffres, éventuellement précédée de "–". Une suite de chiffres précédée de "+" n'est pas considérée comme un nombre.
- Ça peut être un atome non-numérique, toutes les combinaisons de caractères sont bonnes, sauf les séparateurs.

SÉPARATEURS : "espace", ".", "(", ")", ";". On peut constituer toutefois des atomes comprenant des séparateurs grâce à la fonction READCH.

COMMENTAIRES : tout ce qui se trouve entre deux ";" sera ignoré, quoique imprimé si c'est sur cartes.

NOMBRES : LISP 510 n'utilise que les nombres entiers. Approximativement, ils doivent être inclus dans l'intervalle [–70 000, +70000]. Les atomes numériques ne comportent ni C, ni P-valeurs ; ils sont stockés directement dans les listes ou dans les variables.

- INTRODUCTION -

Le système LISP 510 est conçu pour la recherche et l'enseignement en Informatique musicale et en Intelligence Artificielle. Sa facilité d'emploi et sa clarté en font un excellent outil d'enseignement. L'incorporation de fonctions puissantes, et sa possibilité d'emploi en mode conversationnel en font, malgré les limitations en taille mémoire, l'outil le plus utilisé pour la recherche au sein de l'institut d'Intelligence Artificielle de Vincennes. Toutefois, en raison même de sa souplesse, le système est plutôt fragile et vulnérable aux utilisateurs hardis. Qu'ils ne s'en prennent qu'à eux-mêmes, le système étant suffisamment puissant pour permettre une programmation efficiente, (presque) dénuée d'"astuces" (que je déteste).

Ce manuel n'est pas réellement destiné aux débutants, mais aux utilisateurs ayant déjà une certaine expérience de LISP. Il décrit en principe tous les aspects courants du système. Ce manuel n'engage l'auteur que sur les principes de base, et sera sujet à des remaniements, au fur et à mesure de l'évolution normale du système et de ses utilisateurs.

Si, au cours d'un travail, vous détectez une erreur manifeste du système, ou une situation très anomalique, laissez tout dans l'état, et venez me chercher, ça permettra de faire sauter les erreurs qui subsistent (inévitablement) dans le système.

Je crois à propos de remercier J. CHAILLØUX, Giuseppe ENGLERT, Harald ERTZ, J.C. HALGAND, P.L. NEUMAN, J. PØINDRØN, J.E. SCHØETTL qui ont lu la version préparatoire du manuel et dont les excellentes suggestions ont et auront influencé la conception du système dont ce manuel est le manuel.

exemple : redéfinition de SETQ pour qu'elle imprime à l'appel :
" nom de la variable = sa valeur "

```
(DF SETQ (1L)
    (SET (PRIN1 (CAR 1L)) (PRØGN
        (PRIN1 '=) (PRINT (EVAL (CADR 1L))) )))
```

L'appel : (SETQ X 5) provoquera alors l'impression
" X = 5 "

Je puis annuler cette redéfinition en évaluant (RPLACD 'SETQ NIL).

MACRØ-CARACTÈRES :

Il y a la possibilité à la lecture d'appeler une fonction LISP à la seule occurence d'un caractère. Le résultat de l'appel de cette fonction sera substitué à la place du caractère lu. On procède ainsi :

(MACRØ caractère fonction) : ceci dit que désormais, la fonction sera appelée avec NIL comme argument, si le caractère est lu.
Le caractère " ' " est une macro standard. Si elle ne l'était pas, on aurait défini :

(MACRØ ' (LAMBDA () (LIST QUØTE (READ)))).

Et, à la présentation de 'ob, ob étant un atome ou une liste, sera substitué : (QUØTE ob).
Le macro-caractère ainsi défini est alors considéré comme un nouveau séparateur.

ATTENTIØN : un macro-caractère particulier ne peut être défini qu'une seule fois.

S-EXPRESSIØNS PØINTÉES :

Elles sont acceptées par le système, sous la forme la plus générale
(expression . expression)

exemple : .((A.(B . C)) . (D E . (F G H . I)))

A l'impression, sauf modification (voir MØT DE CØNTRØLE) tous les objets édités (par PRIN1 ou PRINT) seront précédés automatiquement d'un espace.

Sauf modification (voir PRELUDE), le nombre d'atomes définis par l'utilisateur ne pourra dépasser 100 (nombres non-inclus).

Sauf modification, l'utilisateur dispose de 1750 doublets libres.

L'utilisateur peut savoir à tout instant combien il reste de doublets libres en évaluant

(CDR 'REM).

FØRME INTERNE DES ATØMES :

Un atome non-numérique est (approximativement) représenté en mémoire ainsi :

C-valeur	P-valeur
PNAME	

La partie PNAME contient la suite de caractères qui représente son nom externe (imprimable).

ATTENTIØN : A la lecture SEULS LES 6 1ERS CARACTERES D'UN ATØME SERØNT PRIS EN CØMPTE, le reste sera ignoré.

La partie C-valeur contiendra, à tout instant, la valeur de l'atome. La partie C-valeur sera considérée comme le CAR de l'atome. Les C-valeurs des atomes définis par l'utilisateur seront initialisées à la valeur "indéfini", ce qui provoquera à l'évaluation l'erreur A8. La C-valeur des SUBRs contient l'adresse du code-machine correspondant.

Certains atomes du système contiennent leur propre adresse en C-valeur, c'est le cas de : NIL, QUØTE, LAMBDA, EXPR, FEXPR, T. Nul besoin, donc de les quoter.

exemple : (LIST NIL QUØTE LAMBDA EXPR FEXPR T)

donnera :→ (NIL QUØTE LAMBDA EXPR FEXPR T)

La partie P-valeur contiendra la P-liste de l'atome. Elle sera considérée comme le CDR de l'atome. Les fonctions DE et DF ne touchent pas aux C-valeurs des atomes. On peut donc redéfinir momentanément une fonction standard comme une EXPR, une NEXPR, ou une FEXPR (voir TYPES DE FØNCTIØNS).

- PRELUDE -

le prélude est un morceau de programme qui permet de modifier certaines caractéristiques du système LISP 510. Quand on charge LISP, on se trouve d'abord dans le prélude, qui se manifeste en tapant "?" à la MAE. On tape alors à la suite du "?" une des commandes suivantes : LIS, PRI, REA, SEE, MØD, CØD, ØCT, DEC, et on va à la ligne. En voici la signification :

LIS : passe la main à LISP proprement dit. Le prélude est alors perdu, on ne peut plus y rentrer.

PRI : imprime sur la MAE un nombre quelconque de cartes quelconques jusqu'à l'arrivée d'une carte

1 2 3
NIL

qui fait passer la main au prélude qui demande une nouvelle commande.

DEC : on tape 6 chiffres octaux suivis d'un espace, et on va à la ligne. Le prélude imprime la version décimale, et attend un nouveau nombre octal. Pour retrouver le prélude, taper un caractère qui n'est pas un chiffre.

ØCT : id. à DEC, mais fait une conversion décimal → octal. Nombres négatifs permis.

REA : lit des cartes de la forme :

adresse octale sur 6 chiffres

en nombre quelconque {
valeur octale
⋮
valeur octale
}

1 2 3
NIL
ou
carte vierge

EVALUATIØN DES VARIABLES :

Ya pas de A-liste. A tout moment, la valeur des variables est accessible dans la C-valeur. A l'entrée de PRØGs ou à l'appel de fonctions dont la variable est argument formel, la valeur d'une variable est empilée dans la pile de travail et restituée à la sortie (voir LISTING DE L'INTERPRETE) au moyen des fonctions BIND et UNBIND.
Ce mode d'acces est tres efficient, mais ne permet plus l'usage des FUNARG comme en LISP 1.5. Reste que nous n'avons pratiquement jamais eu d'occasion sérieuse de les utiliser.

Un bon ensemble de traces, d'éditeurs, de pretty-print, et d'indexs-catalogues est disponible, écrit directement en LISP.

le systeme effectue pour son compte le strict minimum de CØNS. Les garbage-collections dus au systeme seront donc relativement peu fréquents.

Le systeme est défini indépendemment de la CAE 510, et peut donc être transféré rapidement sur un autre ordinateur.



Je peux modifier ces caractéristiques par la commande MØD.

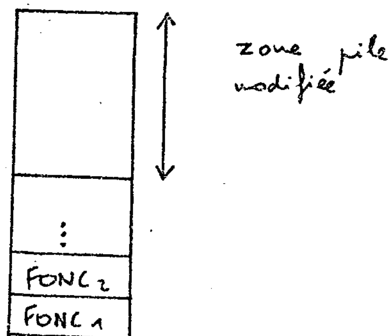
exemple : je tape MØD ; à la ligne ; puis
6⌐⌐ 80⌐ ; à la ligne. Ça fixe le nombre de caractères par ligne à 80.
Puis je tape :
3⌐⌐ 6993 ⌐
4⌐⌐ 6995 ⌐

Ça colle respectivement TØP-SEX à 6993 et BØT-STACK à 6995, me faisant ainsi gagner 15 doublets de plus, pris sur la zone pile.

Quand j'ai assez modifié, je tape un caractère non-numérique, ce qui redonne la main au prélude.

ATTENTIØN : je dois toujours modifier ensemble :
- les adresses de TØP-ATØM et de BØT-SEX d'un multiple de 4 (il faut 4 mots par atome).
- les adresses de TØP-SEX et de BØT-STACK d'un multiple de 2 (il faut 2 mots par doublet).

Enfin la commande CØD permet de rentrer sur cartes de nouvelles SUBRs ou FSUBRs en langage-machine assemblé CAE 510. Ces fonctions seront ainsi stockées :



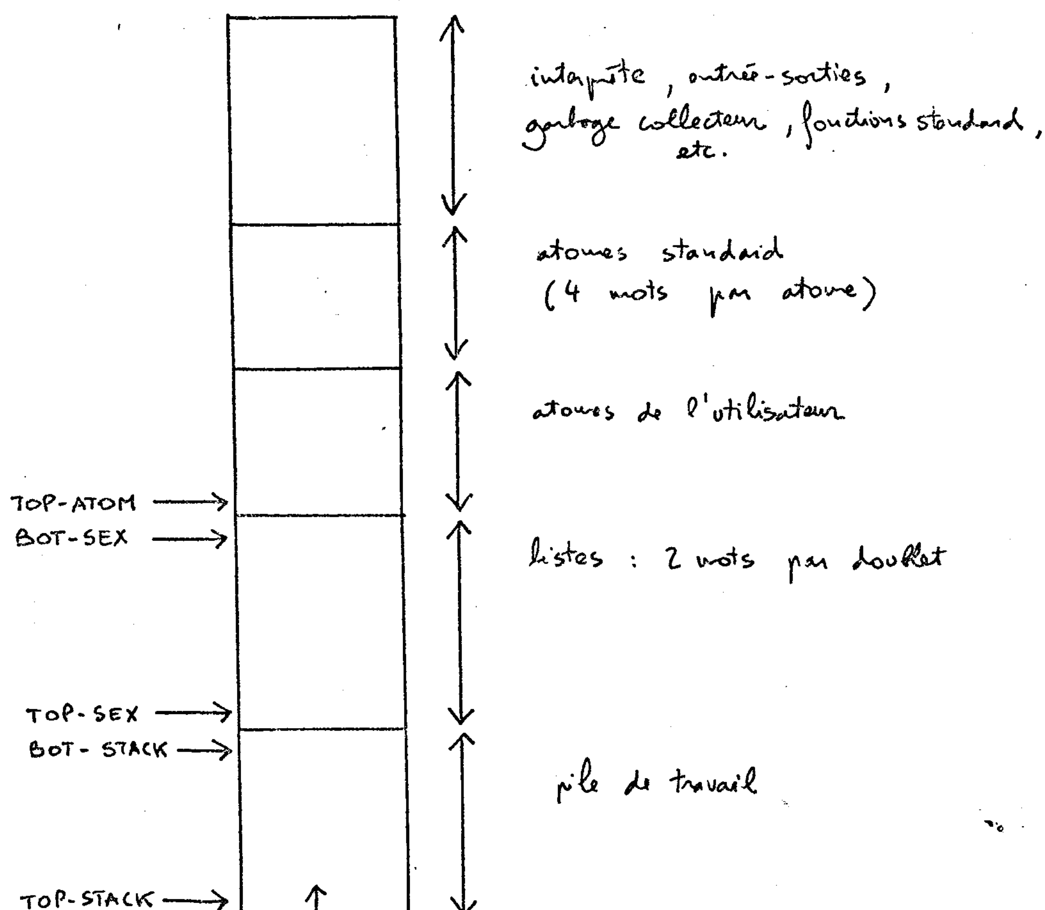
Ces fonctions seront alors considérées comme standard et conservées (même si on évolue (RESET)). Sur cartes, ces fonctions seront ainsi écrites (une ligne par carte).

nom (complété à 6 caractères par ⌐ 8 0 ⌐ en perforation multiple.
nombre d'instructions (en décimal)
SUBR ou FSUBR. Cette carte pourra être omise si le nom est celui d'une fonction déjà existante.
nombre octal (6 caractères) : c'est une instruction assemblée.
⋮
*** {+/-} nb décimal : saut
nom (complété) {+/-} nb décimal : nom de fonction déjà entrée par CØD. Ce nom est l'adresse de sa 1ère instruction
⋮
NIL ou carte vierge : mêmes conventions que pour REA.

A partir de l'adresse en mémoire, ça place consécutivement les valeurs qui suivent.
Une carte NIL pour finir renvoie au prélude.
Une carte vierge provoque la lecture d'une nouvelle série
adresse, val 1, ..., val n.

• SEE, MØD et CØD :

Voici (approximativement) la configuration de la mémoire, après chargement du système.



les noms ridicules désignent les adresses de frontière entre zones.

Si j'envoie au prélude la commande SEE, ça imprime :

1. TØP-ATØM   adresse décimale
2. BØT-SEX   id.
3. TØP-SEX   id.
4. BØT-STACK   id.
5. TØP-STACK   id.
6. LEN-BUF   nombre de caractères imprimables par ligne.

- TYPES DES FØNCTIØNS -

Le système comporte 6 types de fonctions : SUBR, NSUBR, FSUBR, EXPR, NEXPR, FEXPR.
Les SUBR sont des fonctions standard en langage-machine.
Les EXPR sont des fonctions définies en LISP par l'utilisateur.

SUBR, EXPR : les arguments, en nombre fixé sont évalués, puis sont passés à la fonction.

NSUBR, NEXPR : les arguments, en nombre quelconque, sont évalués. Puis ces valeurs sont rassemblées en une liste qui est passée à la fonction.

FSUBR, FEXPR : les arguments, en nombre quelconque, ne sont pas évalués, mais sont rassemblés en une liste qui est passée à la fonction.

Si une fonction (standard ou utilisateur) n'a pas assez d'arguments fournis à l'appel, les arguments manquants sont supposés être NIL. La fonction ne peut donc pas distinguer entre NIL donné comme argument, et pas d'argument du tout. C'est fort commode.
exemple : (FØØ) et (FØØ NIL) reviennent au même.
(SETQ X NIL) a le même effet que (SETQ X).

Si l'appel de la fonction comporte trop d'arguments (pour les EXPR et SUBR), ils sont évalués mais ignorés par la fonction. Donc, si on a une fonction f à n arguments, et qu'on évalue (f x_1 x_2 ... x_m), avec $m > n$, les arguments $x_{n+1}, x_{n+2}, \ldots, x_m$ seront ignorés.
Les arguments formels en trop jouent donc le même rôle que les variables locales des PRØG, et sont initialisés à NIL.

On dispose d'une fonction standard PRØGN, à nombre quelconque d'arguments. Celle-ci évalue ses arguments de gauche à droite et retourne la valeur du dernier.

Les LAMBDA-expressions ont un PRØGN implicite, i.e.

(LAMBDA (A1 A2) e1 e2 .... en)

sera interprété comme :

(LAMBDA (A1 A2) (PRØGN e1 e2 ... en)

et la valeur ramenée sera celle de en.

Une session avec le PRÉLUDE.

```
.
 ?SEE
1. TOP-ATOM     3361
2. BOT-SEX      3365
3. TOP-SEX      6963
4. BOT-STACK    6965
5. TOP-STACK    8191
6. LEN-BUF      72
?MOD
1  3561
2  3565
.
?SEE
1. TOP-ATOM     3561
2. BOT-SEX      3565
3. TOP-SEX      6963
4. BOT-STACK    6965
5. TOP-STACK    8191
6. LEN-BUF      72
?MOD
6  80
.
?SEE
1. TOP-ATOM     3561
2. BOT-SEX      3565
3. TOP-SEX      6963
4. BOT-STACK    6965
5. TOP-STACK    8191
6. LEN-BUF      80
?OCT
64
000100
-1
777777
8191
017777
-4096
770000
.
?DEC
000100
64
777777
-1
017777
8191
770000
-4096
.
?PRI
JACQUES LONCHAMPT(1953-2007) OP.IV
1        'DEB'
888      'ENT'Z;
2        'ENT'I,AL,EKU,C,D,J,GLOK;
3       'ENT''TAB'MEL.(1:2,1:12).,BUF.(1:70).;
4        'ENT''PRO'MOD(X,Y);'VAL'X,Y;'ENT'X,Y;
5         MOD:=X-X%Y*Y;
?LIS
```

LISP.510.2

- MOT DE CONTRÔLE -

Il existe un mot de 18 bits qui contrôle certains états du système. Ce mot est accessible au moyen des fonctions SETBIT et CLRBIT.

BITS	FONCTION DU POSITIONNEMENT	
18	= 1	: LISP n'imprime plus le résultat de l'évaluation à la boucle de lecture. Quelque chose comme ça : (WHILE T (EVAL (READ)))
	= 0	: imprime le résultat de l'évaluation.
17	= 1	: un espace est placé AVANT tout objet, atome ou liste, imprimé dans une ligne.
	= 0	: suppression de cet espace.
16	= 1	: LISP ne tient plus compte, à la lecture, des macro-caractères, " ' " inclus.
	= 0	: en tient compte.
1	= 1	: lectures sur cartes.
	= 0	: au clavier de la MAE.

Au chargement de LISP on a :

18	17	16	...	1
0	1	0		0

Toute erreur (voir ERREURS) redonne au mot de contrôle, la même configuration qu'au chargement.

- FØNCTIØNS DE CØNTRØLE -

(EVAL ob) → évalue ob et ramène sa valeur.
SUBR

(APPLY f l) → applique la fonction f à la liste l d'arguments évalués. Ramène la valeur résultante de cette application. La fonction f ne doit pas être une FSUBR ni une FEXPR.
SUBR

(PRØGN e1 e2 ... en) → évalue en séquence les expressions e1, ..., en et ramène en valeur, la valeur de en.
FSUBR

(EPRØGN l) → évalue en séquence les éléments de la liste l, et ramène en valeur celle du dernier élément de l.
SUBR

(EVLIS l) → l étant une liste (e1 e2 ... en), EVLIS ramène en valeur
SUBR

(val [e1] val [e2] ... val [en])

(AND e1 e2 ... en) → les expressions e1 e2 ... en sont évaluées de gauche à droite jusqu'à ce que la valeur d'un des ei soit NIL : dans ce cas, la valeur du AND est NIL. Si aucun des ei ne donne NIL, la valeur du AND est celle de en.
NSUBR

(ØR e1 e2 ... en) → les expressions e1 e2 ... en sont évaluées de gauche à droite. La valeur du ØR est celle du premier ei dont la valeur n'est pas NIL. Si tous les ei sont NIL, la valeur du ØR est NIL.
NSUBR

(WHILE e1 e2 ... en) → les expressions e1 e2 ... en sont évaluées de gauche à droite tant que la valeur de e1 n'est pas NIL. WHILE boucle ainsi jusqu'à ce que la valeur de e1 soit NIL. WHILE ramène alors la valeur NIL.
FSUBR

- ERREURS -

Si ya une erreur detectée, LISP tape (en rouge) "*ER type" et repasse la main au clavier et à la boucle de lecture de l'interprète. Aucune définition de fonction, ou attribution de valeur à une variable n'est modifiée, et une garbage-collection est effectuée dans la foulée.

Voici les types des erreurs :

LC : erreur de lecture. Ça peut être une liste commençant par ")" ou bien "." non suivi de {liste / atome} ")". En général, ça detecte trop de ")".

RT : l'utilisateur a tenté de faire un RETURN sans être dans un PRØG.

FS : la pile de travail est débordée. Erreur en général due à une récursion infinie.

FM : ya plus de doublets disponibles en zone liste.

AT : ya trop d'atomes définis par l'utilisateur.

A2 : fonction non-définie, detectée dans APPLY. Le nom de cette fonction inconnue est IMPRIMÉ avant le diagnostic.

A6 : GØ ou GØTØ à une étiquette absente du PRØG courant. Le nom de l'étiquette est imprimé.

A8 : tentative d'évaluation d'une variable non-initialisée. Le nom de la variable est imprimé. En général, la variable a été simplement oubliée.

A9 : fonction non-définie, detectée dans EVAL. Le nom de la fonction inconnue est imprimé.

(CØND c_1 c_2 ... c_K) →
FSUBR

CØND comporte un nombre quelconques d'arguments c_1 c_2 ... c_K appelés <u>clauses</u>. Chaque clause c_i est une liste : (e_{1i} e_{2i} ... e_{ni}) de $n \geq 1$ expressions. Les clauses de CØND sont traitées en séquence comme suit :
La 1ere expression e_{1i} est évaluée, ou bien comme <u>fausse</u> si = NIL, ou bien comme <u>vraie</u> si ≠ NIL.
Si la valeur de e_{i1} est vraie, les expressions e_{2i} ... e_{ni} qui suivent dans la clause c_i sont évaluées en séquence, et la valeur du CØND est la valeur de e_{ni}, la dernière expression de la clause. En particulier, si $n = 1$ i.e. si la clause c_i ne comporte qu'<u>une</u> expression, la valeur du CØND est alors celle de e_{1i}.
Si e_{1i} est fausse, alors le reste de la clause c_i est ignoré, et la clause suivante c_{i+1} est traitée.
Si aucun e_{1i} n'est vrai dans aucune clause, la valeur du CØND est NIL.

J'ai emprunté l'explication qui précède au manuel du LISP BBN (Bolt, Beranek, Newman), en raison de sa remarquable clarté.

(PRØG l e_1 e_2 ... e_n) →
FSUBR

l : liste de variables locales, ou NIL si y en a pas.
e_1 ... e_n : suite d'expressions dont chacune peut être précédée d'un atome qui servira d'étiquette aux GØ et GØTØ.
A l'entrée du PRØG, les variables locales sont reservées et initializées à NIL. Puis les e_1 ... e_n sont évaluées en séquence, les étiquettes étant sautées. S'il n'y a pas de RETURN évalués, la valeur du PRØG est celle de e_n.

Les étiquettes numériques sont tout à fait possibles (utiles pour les aiguillages), ainsi que les étiquettes multiples (plusieurs étiquettes consécutives), précédant immédiatement une expression dans un PRØG.

(RETURN ob) → sort du PRØG courant en ramenant la valeur de ob en valeur.
SUBR

(GØ e) →
FSUBR

e : atome qui doit être une étiquette du PRØG courant. GØ fait reprendre l'évaluation du PRØG à partir de l'expression qui suit son étiquette.

(GØTØ e) →
SUBR

id. à GØ, mais l'expression e est évaluée, et doit donc ramener en valeur un atome qui est une étiquette du PRØG courant.

ATTENTIØN : GØ, GØTØ et RETURN peuvent tout à fait être évalués dans des fonctions appelées par un PRØG. Toutefois, sauf à utiliser un ESCAPE, on ne peut sortir d'un PRØG que par un RETURN, ou par la fin du PRØG, ou encore par une erreur (!).

(QUØTE ob) →
FSUBR

ramène ob sans l'évaluer. En LISP 510, on écrit plutôt 'ob.
Si QUØTE n'était pas une fonction standard, on la définirait ainsi :

(DF QUØTE (L) (CAR L))

(ESCAPE f e1 ... en) →
FSUBR

f : un atome non-numérique
e1 ... en : une suite d'expressions à évaluer.
ESCAPE évalue les e1 ... en en séquence. Si dans la portée du ESCAPE se produit l'évaluation de (f ob1 ... obm), on évalue les ob1 ... obm en séquence, et on sort du ESCAPE en ramenant en valeur celle de obm. Si cela ne se produit pas, la valeur du ESCAPE est celle de en.

NØTER : ESCAPE est la fonction de saut la plus puissante compatible avec la structure de contrôle de LISP. Elle n'existe avec cette généralité qu'en LISP 510. Un ESCAPE est une sorte d'étiquette, et sa sortie une sorte de branchement ramenant une valeur.
Dans (ESCAPE EXIT e1 e2 ... en), par exemple, EXIT est lié avec une fonction interne nommée ESC. EXIT devient donc, dans la portée du ESCAPE, une fonction de sortie. Si, au cours de l'évaluation des e1 e2 ... en, on est amené à évaluer (EXIT ob1 ... obm), on évalue les ob1 ... obm et on sort du ESCAPE avec la valeur de obm.
Les ESCAPE peuvent s'imbriquer, et sauter les uns par dessus les autres.

Voici les définitions des fonctions de contrôle, si elles étaient des FEXPR. Elles sont définies en terme de PRØG et d'EVAL.

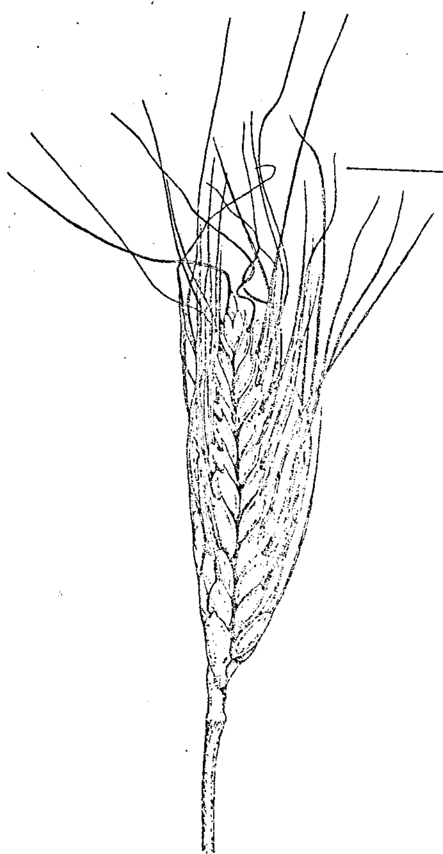
EVAL, APPLY, CØND, ÉVLIS, EPRØGN : voir LISTING DE L'INTERPRETE.

```
(DF PROGN (L) (EPROGN L))

(DF AND (L)
      (PROG (X)
      RE      (COND
                 ((NULL L) (RETURN X)))
              (SETQ X (EVAL (NEXTL L)))
              (COND
                 ((NULL X) (RETURN NIL)))
              (GO RE)))

(DF OR (L)
      (PROG (X)
      RE      (COND
                 ((NULL L) (RETURN NIL)))
              (SETQ X (EVAL (NEXTL L)))
              (COND
                 ((NULL X) (GO RE)))
              (RETURN X)))

(DF WHILE (L)
      (PROG ()
      RE      (COND
                 ((NULL (EVAL (CAR L)))
                  (RETURN NIL)))
              (EPROGN (CDR L))
              (GO RE)))
```



- LISTING DE L'INTERPRETE -

```
(DE EVAL (A1)
      (PROG (CA1 CD1)
             (COND
                ((NULL A1) (RETURN A1))
                ((NUMBP A1) (RETURN A1))
                ((ATOM A1)
                   (COND
                      ((EQ (CAR A1) 'UNDEF) (ERROR A8))
                      ((RETURN (CAR A1))))))
             (SETQ CA1 (CAR A1))
             (SETQ CD1 (CDR A1))
             (AND (EQ CA1 QUOTE) (RETURN (CAR CD1)))
      EVAL03 (COND
                ((NUMBP CA1) (ERROR A9))
                ((ATOM CA1) (GO EVAL5)))
      EVAL4  (COND
                ((MARKED CA1)
                 (UNMARK CA1)
                 (RETURN (MACH CA1 (EVLIS CD1))))
                ((RETURN (APPLY CA1 (EVLIS CD1)))))
      EVAL5  (COND
                ((ATOM (CDR CA1))
                 (SETQ COMM-ESCAPE CA1)
                 (COND
                   ((ISSUBR CA1)
                    (MARK CA1)
                    (GO EVAL4))
                   ((ISFSUBR CA1)
                    (RETURN (EXEC-FSUBR CA1 CD1)))
                   ((EQ (CAR CA1) 'UNDEF) (ERROR A9))
                   (T (SETQ CA1 (CAR CA1))
                      (GO EVAL03))))
                ((EQ (CADR CA1) EXPR)
                 (SETQ CA1 (CADDR CA1))
                 (GO EVAL4))
                ((EQ (CADR CA1) FEXPR)
                 (RETURN (APPLY (CADDR CA1) (CONS CD1 NIL)))))))
```

```
(DE APPLY (A1 A4)
      (PROG ()
      APPLY1 (COND
                 ((ATOM A1)
                  (COND
                     ((NULL A1) (ERROR A2))
                     ((ATOM (CAR A1)) (GO APPLY2))
                     ((EQ (CADR A1) EXPR)
                      (SETQ A1 (CADDR A1))
                      (GO APPLY1))))
                 ((EQ (CAR A1) LAMBDA)
                  (BIND (CADR A1) A4)
                  (SETQ A1 (EPROGN (CDDR A1)))
                  (UNBIND)
                  (RETURN A1))
         (T (SETQ A1 (EVAL A1))
            (GO APPLY1)))
      APPLY2 (COND
                 ((ISSUBR A1) (RETURN (MACH A1 A4)))
                 ((EQ (CAR A1) 'UNDEF) (ERROR A2)))
             (SETQ A1 (CAR A1))
             (GO APPLY1)))


(DE EVLIS (A1)
      (COND
         ((NULL A1) NIL)
         ((CONS (EVAL (CAR A1))
                (EVLIS (CDR A1))))))


(DE EPROGN (A1)
      (WHILE (CDR A1)
             (EVAL (CAR A1))
             (SETQ A1 (CDR A1)))
      (EVAL (CAR A1)))
```

```
(DF COND (A1)
      (PROG (X)
             (WHILE A1
                    (SETQ X (EVAL (CAAR A1)))
                    (AND X (RETURN
                      (COND
                         ((NULL (CDAR A1)) X)
                         ((EPROGN (CDAR A1))))))
                    (SETQ A1 (CDR A1)))))


(DE BIND (X Y)
      (PROG (X1)
             (PUSH LLINK)
             (PUSH SEPARATEUR)
             (COND
                ((NULL X) (GO END))
                ((ATOM X)
                 (PUSH (CAR X))
                 (PUSH X)
                 (RPLACA X Y)
                 (GO END)))
             (WHILE X
                    (SETQ X1 (NEXTL X))
                    (PUSH (CAR X1))
                    (PUSH X1)
                    (RPLACA X1 (NEXTL Y)))
      END    (SETQ LLINK STACKP)))


(DE UNBIND (;; X)
      (WHILE (NEQ (SETQ X (POP)) SEPARATEUR)
             (RPLACA X (POP)))
      (SETQ LLINK (POP)))


(DE PUSH (X) ; EMPILER X ;)


(DE POP (;; X) (SETQ X (CAR STACKP)) ; DEPILER ; X)
```

- FONCTIONS D'ENTREE-SORTIE -

(READ) SUBR → sa valeur est un objet lu, atome ou liste, sur carte ou au clavier de la MAE (voir MØT DE CØNTROLE).

(PRINT ob). SUBR → imprime ob et ramène ob en valeur. L'impression est suivie d'un retour à la ligne, et précédée (sauf modification par (CLRBIT 17)) d'un espace.

(PRIN1 ob) SUBR → id. à PRINT, mais ob est seulement ÉDITÉ dans ligne. Il n'y pas d'impression de la ligne, elle sera après coup imprimée par l'appel d'un PRINT ou d'un TERPRI.

(TERPRI) SUBR → provoque un passage à la ligne (ce qui peut provoquer l'impression d'une ligne éditée), et ramène NIL en valeur.

(TTAB n) SUBR → le prochain objet édité (par PRINT ou PRIN1) sera placé à la position n dans la ligne. TTAB ramène n en valeur.

(SPACES n) SUBR → laisse dans une ligne en cours d'édition, n espaces entre le dernier objet édité et le prochain objet édité dans la ligne. SPACES ramène n en valeur.

ATTENTIØNS :

- si la somme des longueurs des objets édités dans une ligne par des PRIN1 ou des PRINT excède la longueur de ligne standard (voir PRÉLUDE), un retour à la ligne automatique est provoqué par le système, et l'impression se poursuit à la ligne suivante.

- si le lecteur de carte est vide, au cours de la lecture de cartes, on repasse en boucle de lecture clavier MAE en appuyant sur le bouton "END ØF FILE" du lecteur de cartes.

- le contenu des cartes lues est systematiquement imprimé sur la MAE lors de la lecture. Pour inhiber provisoirement l'impression, positionner la clé 17 du pupitre. Enlever la clé 17 pour rétablir l'impression.
- si, au clavier de la MAE, on demande l'évaluation d'un atome, taper après l'atome un espace avant d'aller à la ligne.
- avec (SETBIT 1) tout est lu sur cartes.
  avec (CLRBIT 1) tout est lu au clavier de la MAE.

(READCH n) → $n \in [1, 6]$. READCH rassemble les
SUBR

n caractères immédiatement disponibles dans le buffer de lecture, en un atome, et le ramène en valeur. TOUS les caractères, y compris les séparateurs "(", ")", ".", ";", peuvent être ainsi manipulés.
(READCH) a le même effet que (READCH 1).
Noter que READCH n'effectue aucune conversion, par exemple : (READCH 3)981, ne ramène pas le nombre 981, mais l'atome non-numérique "981".

Des routines d'entrée-sortie sur ruban perforé sont disponibles, et doivent être chargées dans le prélude au moyen de la commande COD. Me contacter à ce sujet.

MUSICIENS ATTENTION : Il existe également une routine de sortie sur ruban perforé en code CAB 500, permettant ainsi d'exploiter à partir de LISP le synthétiseur de son, et l'écran de visualisation.
Un nouveau software en CAB 500 pour le synthétiseur a été conçu et sera entretenu par Jerôme CHAILLOUX. Contactez-le en cas de pépin, ou pour plus de précision.

- FONCTIONS DE DONNÉES -

(ATØM x) SUBR	→	{ T si x est un atome { NIL sinon NIL est, pour ATØM, considéré comme un atome donc (ATØM NIL) → T
(CAR x) SUBR	→	{ si x est un atome : sa valeur { si x est une liste : son 1er élément
(CDR x) SUBR	→	{ si x est un atome : sa P-liste { si x est une liste : x SANS son 1er element
(CAAR x)	→	(CAR (CAR x))
(CADR x)	→	(CAR (CDR x))
(CDAR x)	→	(CDR (CAR x))
(CDDR x)	→	(CDR (CDR x))
(CADDR x) SUBR	→	(CAR (CDDR x))
(EQ x y) SUBR	→	si x et y sont des atomes { T si x = y { NIL sinon si x et y sont des listes { T si x et y sont PHYSIQUEMENT la même liste { NIL sinon
(NEQ x y) SUBR	→	(NØT (EQ x y))
(NULL x) SUBR	→	{ T si x = NIL { NIL sinon
(NØT x) SUBR	→	id. NULL

(CØNS x y) → • si y est une liste, ramène une liste y avec x placé DEVANT l'ancien 1er élément de y
SUBR

• si y est un atome, ramène la paire pointée

(x . y)

(LIST e1 ... en) → ramène la liste des valeurs des e1 ... en.
NSUBR

(LENGTH l) → ramène le nombre d'éléments de la liste l.
SUBR

(NTH n l) → n : un nombre, et l : une liste.
SUBR

NTH ramène la partie de l qui commence par le nième élément.

par exemple : si n = 1, la valeur est l
si n = 2, la valeur est (cdr l)
si n = 3, la valeur est (CDDR l)
etc.

(MEMQ ob l) → ob : un atome, et l : une liste.
SUBR

MEMQ ramène
{ NIL si ob ∉ l
{ sinon la partie de l qui commence par ob.

par exemple : (MEMQ x3 '(x1 x2 x3 x4)) → (x3 x4)

(SETQ ob e) → donne à l'atome ob la valeur de l'expression e, et ramène celle ci en valeur.
FSUBR

(SET e1 e2) → id. à SETQ, mais e1 est évalué.
SUBR

A la valeur ramenée près, l'effet est équivalent à celui de RPLACA.

(NCØNC l1 l2) → concatène physiquement l1 et l2 (qui doivent être des listes), et ramène l1 en valeur.
SUBR

(NCØNC1 l1 e) → (NCØNC l1 (LIST e))
SUBR

Fort utile pour placer un atome à la fin d'une liste.

(EQUAL e1 e2) → { T si e1 et e2 sont identiques
SUBR { NIL sinon

surtout utilisé pour tester l'égalité de 2 listes.

(MAP l f) → applique la fonction f à la liste l et
SUBR à ses CDR successifs, et ramène NIL en valeur.

(MAPC l f) → applique la fonction f aux CARs successifs
SUBR de la liste l et ramène NIL en valeur.

(RPLACA ob1 ob2) →
SUBR

- si ob1 est une liste, RPLACA élimine son 1er élément et place ob2 à la place
- si ob1 est un atome, RPLACA donne à cet atome la valeur ob2.

Dans les 2 cas, la valeur ramenée par RPLACA est ob1. A la valeur ramenée près, l'effet est équivalent à celui de SET.

(RPLACD ob1 ob2) →
SUBR

- si ob1 est une liste, RPLACD remplace son CDR par ob2.
- si ob1 est un atome, RPLACD remplace sa P-liste par ob2.

Dans les 2 cas, la valeur ramenée par RPLACD est ob1.

(PUT ob v ind) → ob : atome ou liste.
SUBR ind : atome.

- Si ob est un atome, place sur sa P-liste la valeur v sous l'indicateur ind.
- Si ob est une liste, elle est considérée comme une P-liste, et v est placé sous l'indicateur ind.

La valeur ramenée par PUT est ob.

(GET ob ind) → si ob est un atome, on considère alors
SUBR sa P-liste, si ob est une liste, elle-même. GET ramène la valeur trouvée sous l'indicateur ind dans la P-liste ; si IND n'existe pas, GET ramène NIL.

NOTER : En LISP 510, les P-listes ont la structure :
(indic1 val1 indic2 val2 ... indicn valn)

Donc GET et PUT, en testant les indicateurs vont de deux en deux éléments.

(NEXTL l) → l doit être une variable qui a pour valeur une liste.
FSUBR

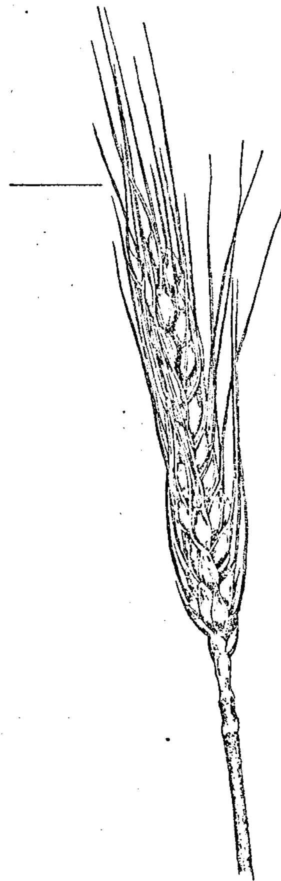
NEXTL ramène le CAR de la valeur de l, et place dans l le CDR de sa valeur.

exemple :

(SETQ X (NEXTL L))

est équivalent à :

(SETQ X (CAR L)) suivi de
(SETQ L (CDR L))



Si les fonctions de données étaient des EXPR, elles seraient définies ainsi :

```
(DE LIST L L)

(DE LENGTH (L)
      (COND
         ((NULL L) 0)
         ((ADD1 (LENGTH (CDR L))))))

(DE NTH (N L)
      (PROG ()
      RE       (COND
                  ((LT N 2) (RETURN L)))
               (SETQ N (SUB1 N))
               (SETQ L (CDR L))
               (GO RE)))

(DE MEMQ (OB L)
      (PROG ()
      RE       (COND
                  ((NULL L) (RETURN NIL))
                  ((EQ (CAR L) OB) (RETURN L)))
               (SETQ L (CDR L))
               (GO RE)))

(DF SETQ (L)
      (SET (CAR L) (EVAL (CADR L))))

(DE EQUAL (E1 E2)
      (COND
         ((ATOM E1) (EQ E1 E2))
         ((EQUAL (NEXTL E1) (NEXTL E2))
          (EQUAL E1 E2))))

(DE MAP (L F)
      (WHILE L
               (F L)
               (SETQ L (CDR L))))

(DE MAPC (L F)
      (WHILE L
               (F (NEXTL L))))
```

```
(DE PUT (OB V IND)
  (COND
      (OB (PROG (X)
                (SETQ X OB)
                (AND (ATOM X) (SETQ X (CDR X)))
      RE        (COND
                   ((EQ (CAR  X) IND)
                    (RPLACA (CDR X) V)
                    (RETURN OB)))
                (SETQ X (CDR X))
                (COND
                   ((NULL (CDR X))
                    (RPLACD X (LIST IND V))
                    (RETURN OB)))
                (SETQ X (CDR X))
                (GO RE)))))


(DE GET (OB IND)
      (PROG ()
                (AND (ATOM OB) (SETQ OB (CDR OB)))
      RE        (COND
                    ((NULL OB) (RETURN NIL))
                    ((EQ (CAR OB) IND)
                     (RETURN (CADR OB))))
                (SETQ OB (CDDR OB))
                (GO RE)))


(DF NEXTL (L)
      (PROG (X)
                (SETQ X (CAR (CAAR L)))
                (SET (CAR L) (CDR (CAAR L)))
                (RETURN X)))
```

- FØNCTIØNS ARITHMÉTIQUES -

(ADD1 x) SUBR	→	$x+1$
(SUB1 x) SUBR	→	$x-1$
(DIFFER x y) SUBR	→	$x-y$
(QUØ x y) SUBR	→	x/y : division entière.
(REM x y) SUBR	→	reste de x/y.
(PLUS x_1 x_2 ... x_n) NSUBR	→	$x_1+x_2+\ldots+x_n$
(TIMES x_1 x_2 ... x_n) NSUBR	→	$x_1 * x_2 * \ldots * x_n$.
(NUMBP x) SUBR	→	{ x si x est un nombre { NIL sinon
(ZERØP x) SUBR	→	{ 0 si $x=0$ { NIL sinon
(LT x y) SUBR	→	{ T si $x<y$ { NIL sinon
(GT x y) SUBR	→	{ T si $x>y$ { NIL sinon
(EQ x y) SUBR	→	{ T si $x=y$ { NIL sinon
(NEQ x y) SUBR	→	{ T si $x \neq y$ { NIL sinon

- FONCTIONS D'ENREGISTREMENT -

(DE at l e1 e2 ... en) →
FSUBR

permet de définir une fonction de nom at et de liste d'arguments l. Cette fonction sera une EXPR. La valeur ramenée par DE est at.

(DF at l e1 e2 ... en) →
FSUBR

id. à DE, mais la fonction définie sera une FEXPR.

- DE place dans la P-liste de at, sous l'indicateur EXPR : (LAMBDA l e1 ... en).
- DF place dans la P-liste de at, sous l'indicateur FEXPR : (LAMBDA l e1 ... en).

REMARQUE : faire (DE at l e1 ... en) est équivalent
à faire : (PUT 'at '(LAMBDA l e1 ... en) EXPR)
et faire (DF at l e1 ... en) est équivalent
à faire : (PUT 'at '(LAMBDA l e1 ... en) FEXPR).

(MACRO c f) →
FSUBR

c : un caractère, et f : une fonction.
Indique que désormais, si le caractère c est lu, la fonction f est appelée avec NIL comme argument. Le résultat de (f NIL) est alors placé dans la donnée lue à la place du caractère c.
La fonction MACRO permet donc de définir des fonctions qui seront appelées A LA LECTURE, par la seule présence du caractère c.

MACRO ramène c en valeur.

- FONCTIONS DU SYSTEME -

(RESET) SUBR → réinitialise le système LISP, et ramène un LISP frais.

(SWITCH n) SUBR → { n si la clé n ∈ [1, 18] du pupitre est positionnée. NIL sinon

MUSICIENS ATTENTION : Le clavier construit par Louis AUDOIRE est testable par SWITCH, et permet donc de construire en LISP un véritable instrument, pour saisir directement des données improvisées.

(OBLIST) SUBR → ramène la liste de tous les atomes non-numériques introduits par l'utilisateur.

(SYSTEM n) SUBR → n : nombre.
SYSTEM ramène l'adresse physique en mémoire n.

exemple d'utilisation de SYSTEM :
L'adresse du pointeur de plus haut atome-utilisateur défini est 36.
Je puis avoir accès à ce pointeur par

(CAR (SYSTEM 36))

et le modifier par :

(SET (SYSTEM 36) qqchose)

exemple : (SETQ SAV (CAR (SYSTEM 36))) : préserve dans SAV les atomes déjà définis. Puis lecture de nouveaux atomes (par exemple en traitement de langage naturel). Si je fais alors (SET (SYSTEM 36) SAV), je restitue l'ancienne valeur, et tous les nouveaux atomes lus sont considérés comme inexistants.

(SETBIT n) SUBR → place à 1 le bit n ∈ [1, 18] du mot de contrôle et ramène n en valeur.

(CLRBIT n) → place à 0 le bit n ∈ [1, 18] du mot de contrôle et ramène n en valeur.

- ADRESSES IMPØRTANTES DU SYSTEME (en décimal) -

16	:	adresse du mot de contrôle.
20	:	contient l'adresse du dernier macro-caractère défini (initialisé à « ? »).
24	:	contient la longueur de ligne standard.
30	:	contient l'adresse du 1er mot de zone liste.
31	:	contient l'adresse du dernier doublet de zone liste.
33	:	contient l'adresse du 1er doublet de la liste libre.
34	:	contient l'adresse du plus haut atome usager possible.
36	:	contient l'adresse du dernier atome-utilisateur défini.
37	:	contient l'adresse du 1er mot de zone pile.
38	:	contient l'adresse du dernier mot de zone pile.
39	:	contient l'adresse du pointeur de pile (STACKP).

ATTENTIØN : si on veut interrompre un traitement en cours, par exemple parce que ça semble boucler, ou pour une autre raison, appuyez sur le bouton du pupitre : "INTERVENTIØN ØPÉRATEUR". Vous vous retrouvez alors en boucle de lecture au clavier, et une garbage-collection est provoquée dans la foulée. Ça a le même effet qu'une erreur, provoquée manuellement (voir ERREURS).

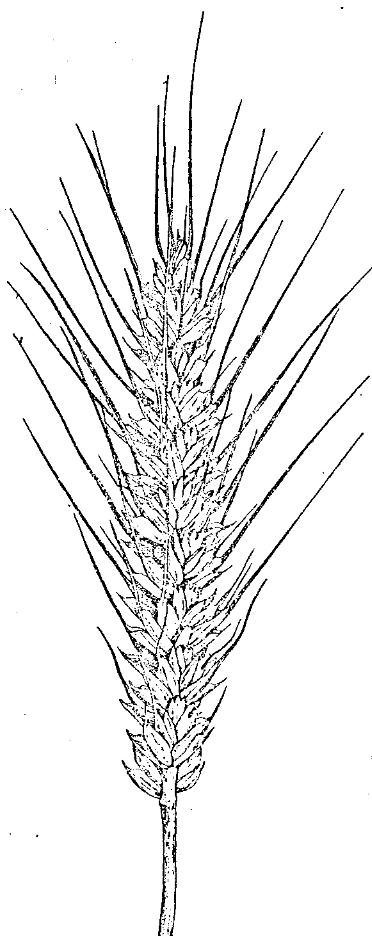
Si vous voulez vous recharger un nouveau système LISP (prélude compris), appuyez deux fois sur la touche "ANALYSE" du pupitre : ça rebobine la bande. Appuyez alors sur MANUEL, ENREGISTREMENT, AUTØ, et vous vous retrouvez dans le prélude.

LISP démarre à l'adresse 100_8, en initialisant les interruptions. En cas de pépin, si on veut se récupérer sur LISP, positionner la clé 7 (et elle seule), appuyez sur MANUEL, ENREGISTREMENT, SW→E, E⟺M, TRANSFERT E, AUTØ (ça revient à faire RESET). Si vous voulez conserver vos définitions, même manœuvre avec les clés 1, 2, 5, 8.

LANGAGE - MACHINISTES ATTENTION :

Un LAP existe, et vous permet donc d'écrire des fonctions directement en logandes, qui seront assemblés en une seule passe, et chargés, permettant ainsi si vous les rentrez au clavier, une mise au point conversationelle. On peut faire communiquer des fonctions LAP - LAP, LAB - SUBR, LAP - FSUBR, LAP - LISP. Pour en savoir plus, lire la brochure "LAP 510".

Un compilateur (exécution ≃ 30 fois plus rapide) est en cours de mise au point et sera décrit séparément.



- LISTING DU PROGRAMME DE TRACE -

```
(SETQ TRACE (READCH))

(SETQ PRINTL (READCH 6))-----)
(SETQ PRTRAC (READCH 6))(-----

(DE GETD (%L)
   (OR (GET %L EXPR) (GET %L FEXPR)))

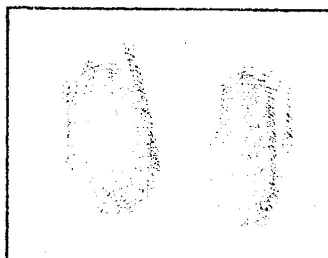
(DE PUTD (%X1 %X2)
  (COND
   ((GET %X1 EXPR) (PUT %X1 %X2 EXPR))
   ((GET %X1 FEXPR) (PUT %X1 %X2 FEXPR)) ))

(DF TRACE (%L)
  (MAPC %L  '(LAMBDA (%X1)
      (SETQ %X2 (GETD %X1))
      (OR (GET %X1 'PRINTL) (PUT %X1 %X2 'PRINTL))
      (PUTD %X1 (LIST (CAR %X2) (CADR %X2)
                     (LIST 'PRTRAC (CADR %X2) %X1) )) )))

(DF PRTRAC (%L %X1 %X2)
    (SETQ %X1 (CAR %L))
    (SETQ %X2 (CADR %L))
    (PRINTL PRINTL (COND ((ATOM %X1) (EVAL %X1))
                         (T (EVLIS %X1)) ))
    (SETQ %X1 (EVAL (CADDR (GET %X2 'PRINTL) )))
    (PRINTL PRTRAC %X1)
    %X1)

(DE PRINTL (%X1 %X3)
  (PRIN1 %X1) (PRIN1 %X2) (PRIN1 TRACE) (PRINT %X3))

(DF UNTRAC (%L)
      (MAPC %L
         '(LAMBDA (%L)
          (RPLACA (CDDR %L) (GET %L 'PRINTL))
          (RPLACD (CDDR %L)) )))
```



```
 (DE REVERSE (L1 L2)
?    (COND
?       ((NULL L1) L2)
?       ((REVERSE (CDR L1)
?                 (CONS (CAR L1) L2)))))
 REVERS
?
 (REVERSE '(A B C D E))
 (E D C B A)
?
 (TRACE REVERSE)
 NIL
?
 (REVERSE '(A B C D E))
 -----) REVERS   ((A B C D E) NIL)
 -----) REVERS   ((B C D E) (A))
 -----) REVERS   ((C D E) (B A))
 -----) REVERS   ((D E) (C B A))
 -----) REVERS   ((E) (D C B A))
 -----) REVERS   (NIL (E D C B A))
 (----- REVERS   (E D C B A)
 (----- REVERS   (E D C B A)
 (----- REVERS   (E D C B A)
 (----- REVERS   (E D C B A)
 (----- REVERS   (E D C B A)
 (----- REVERS   (E D C B A)
 (E D C B A)
?
 (UNTRAC REVERSE)
 NIL
?
 (REVERSE '(JE CHASSE LE SNARK))
 (SNARK LE CHASSE JE)
?
```

- EXEMPLE D'UTILISATION DU PROGRAMME DE TRACE -

En général, on tracera les fonctions $f_1\ f_2 \ldots f_n$, en appelant (TRACE $f_1\ f_2 \ldots f_n$), id. pour UNTRAC qui supprime la trace.

A l'entrée d'une fonction tracée est imprimé :
"-----)" puis son nom, puis la liste de ses arguments,
et à la sortie :
"(-----" puis son nom, puis la valeur ramenée par cet appel.

- LISTING DU PRETTY-PRINT -

```
(DF PRETTY (L I LESCAPE XPRTTY)
   (CLRBIT 17)
   (SETQ XPRTTY '(LAMBDA ()  (SETQ I (ADD1 I))
                             (MAPC E '(LAMBDA (E)
                               (ENDLINE) (SUPERPRINT E) ))
                             (SETQ I (SUB1 I))
                             (PRIN1 RPAR) ))
   (MAPC L
      '(LAMBDA (L) (TERPRI) (PRIN1 LPAR) (PRINT  L)
                   (TTAB (SETQ I 3))
                   (SUPERPRINT (CADDR L)) (PRIN1 RPAR) ))
       (TERPRI))


(DE ENDLINE () (TERPRI) (TTAB (TIMES I 3)))


(SETQ LPAR (READCH))(
(SETQ RPAR (READCH)))
```

Le programme de PRETTY-PRINT permet de mettre en page des fonctions enregistrées, de la façon la plus lisible possible.
Il se compose des 3 fonctions PRETTY, ENDLINE et SUPERPRINT.
Il va de soi que PRETTY-PRINT supprime tous les commentaires, les macro-caractères, et imprime les atomes tels qu'ils ont été effectivement enregistrés, i.e. ne comportant pas plus de 6 caractères.

En général, on demandera la mise en page successive des fonctions f1, f2, ... fn en évaluant :

(PRETTY f1 f2 ... fn)

PRETTY est fort rapide, et fait le minimum de CONS.

```
(DE SUPERPRINT (E L)
   (ESCAPE EXIT
    (COND
     ((ATOM E) (PRIN1 E))
     (T
      (PRIN1 LPAR)
      (WHILE E (SETQ L (NEXTL E))
         (COND
          ((NULL E))
          ((OR (MEMQ L LESCAPE)
               (MEMQ L '(OR AND LIST PLUS TIMES WHILE PROGN MAP MAPC)))
                  (PRIN1 L)
           (EXIT (XPRTTY)))
          ((EQ L 'ESCAPE)      (PRIN1 L)
           (SETQ LESCAPE (CONS (CAR E) LESCAPE))
           (EXIT (XPRTTY) (NEXTL LESCAPE)))
          ((EQ L LAMBDA)
           (PRIN1 LAMBDA) (SPACES 1)
           (SUPERPRINT (NEXTL E)) (EXIT (XPRTTY)) )
          ((EK L 'COND)
           (PRIN1 L) (SETQ I (ADD1 I))
           (MAPC E
            '(LAMBDA (E) (ENDLINE) (PRIN1 LPAR)
                         (SUPERPRINT (NEXTL E))
             (XPRTTY) ))
           (SETQ I (SUB1 I))
           (EXIT (PRIN1 RPAR)) )
          ((EQ L 'PROG)
           (PRIN1 L) (SPACES 1) (SUPERPRINT (NEXTL E))
           (WHILE E
            (ENDLINE)
            (COND
             ((ATOM (CAR E))
              (PRIN1 (NEXTL E)) (TTAB (TIMES 3 (PLUS I 2))) )
             (T (SPACES 6)))
            (SETQ I (PLUS I 2)) (SUPERPRINT (NEXTL E))
            (SETQ I (DIFFER I 2)))
           (SETQ I (SUB1 I))
           (EXIT (PRIN1 RPAR)) ))
       (SUPERPRINT L) (AND E (SPACES 1)))
      (PRIN1 RPAR)) )))
```

```
(PRETTY PRETTY ENDLINE)

(PRETTY
  (LAMBDA (L I LESCAP XPRTTY)
          (CLRBIT 17)
          (SETQ XPRTTY (QUOTE (LAMBDA NIL
             (SETQ I (ADD1 I))
             (MAPC
                E
                (QUOTE (LAMBDA (E)
                   (ENDLIN)
                   (SUPERP E))))
             (SETQ I (SUB1 I))
             (PRIN1 RPAR))))
          (MAPC
             L
             (QUOTE (LAMBDA (L)
                (TERPRI)
                (PRIN1 LPAR)
                (PRINT L)
                (TTAB (SETQ I 3))
                (SUPERP (CADDR L))
                (PRIN1 RPAR))))
          (TERPRI)))
(ENDLIN
  (LAMBDA NIL
          (TERPRI)
          (TTAB (TIMES
             I
             3))))
```

Exemple d'utilisation de PRETTY-PRINT. Appliqué à PRETTY et ENDLINE.

Les 2 pages qui suivent montrent encore l'effet de PRETTY, sur deux versions de procédures d'unification en démonstration automatique de théorème.
La 1ère est de Chin-Liang CHANG et Richard Char-Tung LEE,
la 2de est de R. WALDINGER et K.N. LEVITT.

```
(PRETTY UNIFY)

(UNIFY
  (LAMBDA (E1 E2)
          (PROG (D U D1 D2)
                (AND
                   (NEQ (LENGTH E1) (LENGTH E2))
                   (RETURN (QUOTE NOT)))
          B1    (SETQ D (DISAGR E1 E2))
                (OR
                   D
                   (RETURN (REVERS U)))
                (SETQ D1 (CAR D))
                (SETQ D2 (CADR D))
                (COND
                   ((OR
                      (MEMQ D1 XLIST)
                      (MEMQ D1 YLIST))
                      (GO B3)))
                (COND
                   ((OR
                      (MEMQ D2 XLIST)
                      (MEMQ D2 YLIST))
                      (GO B4)))
          B2    (RETURN (QUOTE NOT))
          B3    (AND
                   (INSIDE D1 D2)
                   (GO B2))
                (SETQ U (CONS D U))
                (SETQ E1 (SUBST D2 D1 E1))
                (SETQ E2 (SUBST D2 D1 E2))
                (GO B1)
          B4    (AND
                   (INSIDE D2 D1)
                   (GO B2))
                (SETQ U (CONS (REVERS D) U))
                (SETQ E1 (SUBST D1 D2 E1))
                (SETQ E2 (SUBST D1 D2 E2))
                (GO B1))))
```

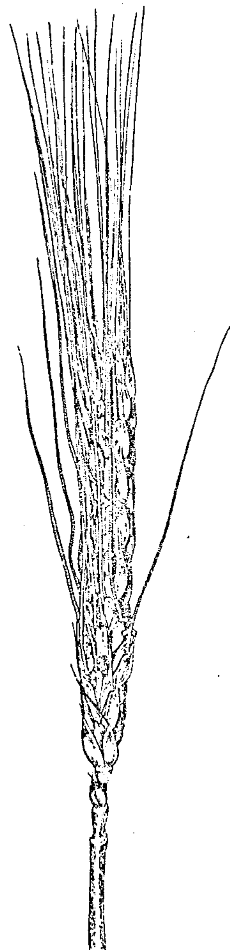
```
(DE UNIFY(X Y M1 M2)(COND((EQ X Y)NIL)((VAR X)(COND
      ((OCCURS X Y) 'NOT)((LIST
 (CONS X Y)))))((VAR Y)(COND((OCCURS Y X)'NOT)((LIST(CONS Y X)))))
 ((ATOM X)'NOT)((ATOM Y)'NOT)(T(SETQ M1 (UNIFY(NEXTL X)(NEXTL Y)))
 (COND((EQ M1 'NOT)'NOT)(T(SETQ M2(UNIFY(VSUBST M1 X)
 (VSUBST M1 Y)))(COND((EQ M2 'NOT)'NOT)((NCONC M2 M1))))))))
 UNIFY
```

```
(PRETTY UNIFY)

(UNIFY
  (LAMBDA (X Y M1 M2)
           (COND
              ((EQ X Y)
                 NIL)
              ((VAR X)
                 (COND
                    ((OCCURS X Y)
                       (QUOTE NOT))
                    ((LIST
                       (CONS X Y)))))
              ((VAR Y)
                 (COND
                    ((OCCURS Y X)
                       (QUOTE NOT))
                    ((LIST
                       (CONS Y X)))))
              ((ATOM X)
                 (QUOTE NOT))
              ((ATOM Y)
                 (QUOTE NOT))
              (T
                 (SETQ M1 (UNIFY (NEXTL X) (NEXTL Y)))
                 (COND
                    ((EQ M1 (QUOTE NOT))
                       (QUOTE NOT))
                    (T
                       (SETQ M2 (UNIFY (VSUBST M1 X) (VSUBST M1 Y)))
                       (COND
                          ((EQ M2 (QUOTE NOT))
                             (QUOTE NOT))
                          ((NCONC M2 M1)))))))))
```

- TEST LISP 510 -

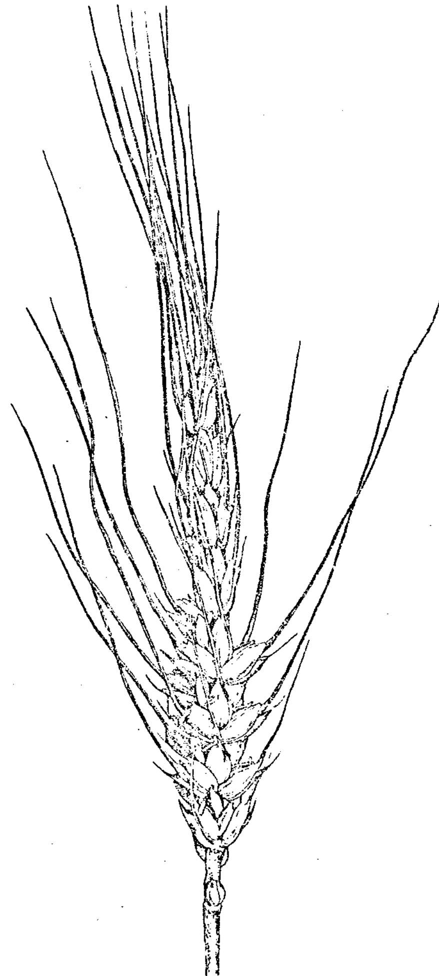
```
(DE TEST() (WHILE T (PRINT (EVAL (READ)))
                    (MAPC '(------ ------ ------) 'PRIN1)
                    (TERPRI)))
 TEST
(TEST)
(SETQ SAVE (CAR (SYSTEM 36 ; NATOMP = 36 ; )))
 SAVE
 ------ ------ ------
; COMMENTAIRE ***; '(COMMEN TAIRE)
 (COMMEN TAIRE)
 ------ ------ ------
NIL
 NIL
 ------ ------ ------
12
 12
 ------ ------ ------
-12
 -12
 ------ ------ ------
'A
 A
 ------ ------ ------
''A
 (QUOTE A)
 ------ ------ ------
'''A
 (QUOTE (QUOTE A))
 ------ ------ ------
()
 NIL
 ------ ------ ------
'(A)
 (A)
 ------ ------ ------
'(A . (B . (C . D)))
 (A B C . D)
 ------ ------ ------
'(((A . B) . C) . D)
 (((A . B) . C) . D)
 ------ ------ ------
(COND (NIL) (T 1))
 1
 ------ ------ ------
(COND (1))
 1
 ------ ------ ------
(COND (T 1 2 3))
 3
 ------ ------ ------
((LAMBDA L L) 'A.'B 'C)
 (A B C)
 ------ ------ ------
(SETQ L1 '((A . B) C D))
 ((A . B) C D)
 ------ ------ ------
```



```
(CAR L1)
 (A . B)
------ ------ ------
(CDR L1)
 (C D)
------ ------ ------
(CDR 'REM)
 1800
------ ------ ------
(CADR L1)
 C
------ ------ ------
(CDAR L1)
 B
------ ------ ------
(CAAR L1)
 A
------ ------ ------
(CDDR L1)
 (D)
------ ------ ------
(CADDR L1)
 D
------ ------ ------
(PROGN 1 2 3 4)
 4
------ ------ ------
(SETQ L2 '(5 6 7 8))
 (5 6 7 8)
------ ------ ------
(EPROGN L2)
 8
------ ------ ------
(EVLIS L2)
 (5 6 7 8)
------ ------ ------
(SET 'L3 '((PRIN1 1) (PRIN1 2) (PRINT 3)))
 ((PRIN1 1) (PRIN1 2) (PRINT 3))
------ ------ ------
(EVLIS L3)
 1 2 3
 (1 2 3)
------ ------ ------
(EPROGN L3)
 1 2 3
 3
------ ------ ------
(EVAL (CONS 'PROGN L3))
 1 2 3
 3
------ ------ ------
```

```
(AND 1 2)
 2
 ------ ------ ------
(AND 1 2 NIL 3)
 NIL
 ------ ------ ------
(AND NIL 1 2)
 NIL
 ------ ------ ------
(OR 1 2)
 1
 ------ ------ ------
(OR 1 2 NIL 3)
 1
 ------ ------ ------
(OR NIL NIL NIL 2)
 2
 ------ ------ ------
(SETQ L4 L2)
 (5 6 7 8)
 ------ ------ ------
(PROGN (WHILE L4 (PRIN1 (NEXTL L4))) (TERPRI))
 5 6 7 8
 NIL
 ------ ------ ------
(MAP L3 '(LAMBDA (X) (PRINT X)))
 ((PRIN1 1) (PRIN1 2) (PRINT 3))
 ((PRIN1 2) (PRINT 3))
 ((PRINT 3))
 NIL
 ------ ------ ------
(MAP L3 '(LAMBDA (X) (EPROGN X)))
 1 2 3
 2 3
 3
 NIL
 ------ ------ ------
(MAPC L3 '(LAMBDA (X) (PRINT X)))
 (PRIN1 1)
 (PRIN1 2)
 (PRINT 3)
 NIL
 ------ ------ ------
(MAPC L3 'EVAL)
 1 2 3
 NIL
 ------ ------ ------
(CONS 'A 'B)
 (A . B)
 ------ ------ ------
(CONS 'A '(B C))
 (A B C)
 ------ ------ ------
(CONS '(A B) 'C)
 ((A B) . C)
 ------ ------ ------
(CONS '(A B) '(C))
 ((A B) C)
 ------ ------ ------
```

```
(EQ 'A 'B)
 NIL
------ ------ ------
(EQ 'A 'A)
 T
------ ------ ------
(EQ 1 1)
 T
------ ------ ------
(EQ '(A) '(A))
 NIL
------ ------ ------
(EQ 0 1)
 NIL
------ ------ ------
(NEQ 'A 'B)
 T
------ ------ ------
(NEQ 'A 'A)
 NIL
------ ------ ------
(NEQ 0 1)
 T
------ ------ ------
(NEQ '(A) '(A))
 T
------ ------ ------
(ADD1 0)
 1
------ ------ ------
((LAMBDA (X) (ADD1 X)) 3)
 4
------ ------ ------
(SUB1 0)
 -1
------ ------ ------
(SUB1 -1)
 -2
------ ------ ------
(NULL 'A)
 NIL
------ ------ ------
(NULL NIL)
 T
------ ------ ------
(NULL)
 T
------ ------ ------
(NOT NIL)
 T
------ ------ ------
(NOT 'A)
 NIL
------ ------ ------
```



```
(ZEROP 0)
 T
 ------ ------ ------
(ZEROP 1)
 NIL
 ------ ------ ------
(LIST 'A 'B 'C)
 (A B C)
 ------ ------ ------
(RPLACA 'A 'B)
 A
 ------ ------ ------
A
 B
 ------ ------ ------
(RPLACD 'A 'C)
 A
 ------ ------ ------
(CDR 'A)
 C
 ------ ------ ------
(CONS (CAR 'A) (CDR 'A))
 (B . C)
 ------ ------ ------
(ATOM 'A)
 T
 ------ ------ ------
(ATOM 1)
 T
 ------ ------ ------
(ATOM NIL)
 T
 ------ ------ ------
(ATOM '(A))
 NIL
 ------ ------ ------
(EQUAL (OBLIST) (OBLIST))
 T
 ------ ------ ------
(EQUAL (OBLIST) (CONS 'X (OBLIST)))
 NIL
 ------ ------ ------
(OBLIST)
 (X L4 L3 L2 L1 L D C B A TAIRE COMMEN SAVE ------ TEST)
 ------ ------ ------
(LENGTH (OBLIST))
 15
 ------ ------ ------
(LENGTH NIL)
 0
 ------ ------ ------
```

```
(NTH 1 (OBLIST))
 (X L4 L3 L2 L1 L D C B A TAIRE COMMEN SAVE ------ TEST)
 ------ ------ ------
(NTH (LENGTH (OBLIST)) (OBLIST))
 (TEST)
 ------ ------ ------
(NCONC '(A B) '(C D))
 (A B C D)
 ------ ------ ------
(NCONC '(A B) 'C)
 (A B . C)
 ------ ------ ------
(NCONC1 '(A B) 'C)
 (A B C)
 ------ ------ ------
(APPLY 'CONS '(A B))
 (A . B)
 ------ ------ ------
(LT 1 2)
 T
 ------ ------ ------
(LT 2 1)
 NIL
 ------ ------ ------
(LT -3 -2)
 T
 ------ ------ ------
(LT -2 -3)
 NIL
 ------ ------ ------
(GT 1 2)
 NIL
 ------ ------ ------
(GT 2 1)
 T
 ------ ------ ------
(GT -3 -2)
 NIL
 ------ ------ ------
(GT -2 -3)
 T
 ------ ------ ------
(DIFFER 5 -5)
 10
 ------ ------ ------
(DIFFER -5 -5)
 0
 ------ ------ ------
(DIFFER 5 5)
 0
 ------ ------ ------
```

```
(NUMBP 5)
 5
------ ------ ------
(NUMBP 'A)
 NIL
------ ------ ------
(NUMBP '(A))
 NIL
------ ------ ------
(DE REVERSE (L M) (WHILE L (SETQ M (CONS (NEXTL L) M))) M)
 REVERS
------ ------ ------
(REVERSE '(A B C))
 (C B A)
------ ------ ------
(REVERSE '((A B) (C D) (E F)))
 ((E F) (C D) (A B))
------ ------ ------
(DF PPRINT (L) (MAPC L 'PRIN1) (TERPRI))
 PPRINT
------ ------ ------
(PPRINT ARGUM ENTS NON EVAL UES)
 ARGUM ENTS NON EVAL UES
 NIL
------ ------ ------
(CDR 'REVERSE)
 (EXPR (LAMBDA (L M) (WHILE L (SETQ M (CONS (NEXTL L) M))) M
))
------ ------ ------
(CDR 'PPRINT)
 (FEXPR (LAMBDA (L) (MAPC L (QUOTE PRIN1)) (TERPRI)))
------ ------ ------
(CDR 'FF)
 NIL
------ ------ ------
(PUT 'FF 1 'UN)
 FF
------ ------ ------
(GET 'FF 'UN)
 1
------ ------ ------
(CDR 'FF)
 (UN 1)
------ ------ ------
(PUT 'FF 2 'DEUX)
 FF
------ ------ ------
(CDR 'FF)
 (UN 1 DEUX 2)
------ ------ ------
(PUT 'FF 'ONE 'UN)
 FF
------ ------ ------
(CDR 'FF)
 (UN ONE DEUX 2)
------ ------ ------
```

```
(READCH)-
 -
 ------ ------ ------
(READCH 1)*
 *
 ------ ------ ------
(READCH 2)..
 ..
 ------ ------ ------
(READCH 3).).
 .).
 ------ ------ ------
(READCH 4)())(
 ())(
 ------ ------ ------
(READCH 6)-----)
 -----)
 ------ ------ ------
(OBLIST)
 (-----) ())( .). .. * - ONE DEUX UN FF UES NON ENTS ARGUM
PPRINT F E M REVERS X L4 L3 L2 L1 L D C B A TAIRE
COMMEN SAVE ------ TEST)
 ------ ------ ------
(SPACES 70)
                                                                              70
 ------ ------ ------
(PLUS 10 10)
 20
 ------ ------ ------
(PLUS 10 10 10)
 30
 ------ ------ ------
(PLUS 10 -10 10)
 10
 ------ ------ ------
(PLUS 10)
 10
 ------ ------ ------
(TIMES 1 2 3 4 5 6 7)
 5040
 ------ ------ ------
(TIMES -1 -2 -3 -4 -5 -6 -7)
 -5040
 ------ ------ ------
(MEMQ 'D '(A B C D E F))
 (D E F)
 ------ ------ ------
(MEMQ 'F '(A B C D E F))
 (F)
 ------ ------ ------
(MEMQ 'G '(A B C D E F))
 NIL
 ------ ------ ------
```

```
(QUO 18 3)
 6
 ------ ------ ------
(QUO -18 3)
 -6
 ------ ------ ------
(QUO 18 -3)
 -6
 ------ ------ ------
(QUO -18 -3)
 6
 ------ ------ ------
(QUO 7 2)
 3
 ------ ------ ------
(REM 3 2)
 1
 ------ ------ ------
(REM 2 3)
 2
 ------ ------ ------
(SETQ X 'RIEN)
 RIEN
 ------ ------ ------
(PROG (X FF)
            (SETQ X 5)
     RE1    (GOTO (SETQ X (SUB1 X)))
     1      (PRINT 'UN) (GO RE1)
     2      (PRINT 'DEUX) (GO RE1)
     3      (PRINT 'TROIS) (GO RE1)
     4      (PRINT 'QUATRE) (GO RE1)
       0 SUITE
            (SETQ X '(A B C D))
            (SETQ FF '(LAMBDA (X) (PRINT X) (GO RE2)))
     RE2    (GOTO (NEXTL X))
     A      (FF 'AA)
     B      (FF 'BB)
     C      (FF 'CC)
     D      (PRINT X) (RETURN 'OK))
 QUATRE
 TROIS
 DEUX
 UN
 AA
 BB
 CC
 NIL
 OK
 ------ ------ ------
X
 RIEN
 ------ ------ ------
```

```
(PPRINT A B C D)
 A B C D
 NIL
 ------ ------ ------
(CLRBIT 17)
17
-----------------
(PPRINT A B C D)
ABCD
NIL
-----------------
(SETBIT 17)
 17
 ------ ------ ------
(SETQ LL NIL)
 NIL
 ------ ------ ------
(DE FORMAT (L NC NAL ;; X NN)
      (SETQ X 1)
      (SETQ NN NAL)
      (WHILE L
             (TTAB X)
             (PRIN1 (NEXTL L))
             (SETQ X (PLUS X NC))
             (SETQ NN (SUB1 NN))
             (COND
                ((ZEROP NN)
                 (TERPRI)
                 (SETQ NN NAL)
                 (SETQ X 1))))
      (OR (EQ NN NAL) (TERPRI))
      T)
 FORMAT
 ------ ------ ------
(RPLACA 'PP '((SETQ X -4) (WHILE
             (NEK (SETQ Y (PLUS NIL (SETQ X (PLUS X 4))))
                  (PLUS 'SYS3 4))
      (SETQ LL (NCONC1 LL Y))) (FORMAT LL 8 9)))
 PP
 ------ ------ ------
(EPROGN PP)
 NIL      REM      QUOTE    '        READ     ADD1     SUB1     ATOM     CAR
 CDR      CAAR     CADR     CDAR     CDDR     CADDR    ZEROP    NEQ      NTH
 LENGTH   SPACES   TTAB     READCH   SYSTEM   NCONC1   RESET    SETBIT   CLRBIT
 NUMBP    NULL     NOT      SWITCH   OBLIST   EVLIS    GOTO     RETURN   PRIN1
 PRINT    TERPRI   EPROGN   EVAL     QUO      DIFFER   EQ       LT       GT
 EQUAL    CONS     MEMQ     GET      RPLACA   RPLACD   NCONC    MAP      MAPC
 SET      APPLY    PUT      LIST     PLUS     TIMES    PROGN    AND      OR
 WHILE    ESCAPE   ESC      PROG     COND     DF       GO       SETQ     MACRO
 NEXTL    DE       LAMBDA   EXPR     FEXPR    T        SYS1     SYS2     SYS3
 T
 ------ ------ ------
(OBLIST)
 (Y PP NN NAL NC FORMAT LL OK CC BB AA RE2 SUITE QUATRE
TROIS RE1 RIEN G ------) ())( .). .. * - ONE DEUX UN FF
UES NON ENTS ARGUM PPRINT F E M REVERS X L4 L3 L2 L1 L D
C B A TAIRE COMMEN SAVE ------ TEST)
 ------ ------ ------
```

```
(SET (SYSTEM 36) SAVE)
 SAVE
------ ------ ------
(OBLIST)
 (SAVE ------ TEST)
------ ------ ------
(MACRO # (LAMBDA () (LIST LAMBDA () (LIST
             'LIST (READ) '(READ) '(READ) ))))
 #
------ ------ ------
(MACRO + #'PLUS)
 +
------ ------ ------
(MACRO * #'TIMES)
 *
------ ------ ------
+ 5 5
 10
------ ------ ------
* 11 23
 253
------ ------ ------
+ * 41 14 * 4 -251
 -430
------ ------ ------
'+ 5 5
 (PLUS 5 5)
------ ------ ------
'* 11 23
 (TIMES 11 23)
------ ------ ------
'+ * 41 14 *4 -251
 (PLUS (TIMES 41 14) (TIMES 4 -251))
------ ------ ------
(OBLIST)
 (* + # SAVE ------ TEST)
------ ------ ------
(SETBIT 16)
 16
------ ------ ------
(SET (SYSTEM 36) SAVE)
 SAVE
------ ------ ------
(SET (SYSTEM 20) (QUOTE '))  ; MACTET = 20 ;
 '
------ ------ ------
(RPLACD (QUOTE '))
 '
------ ------ ------
(CLRBIT 16)
 16
------ ------ ------
(OBLIST)
 (SAVE ------ TEST)
------ ------ ------
(RESET (PRINT '(FIN DU TEST LIST 510-2)))
 (FIN DU TEST LIST 510-2)

?
```

- LISTING DU PRØGRAMME D'INDEXATIØN -

```
(DE DIRECTORY (FN FN1)
   (PRINST FN FN1)
   (MAPC DONE '(LAMBDA (FN) (PRINT (PRINST NIL FN)))))

(DE PRINST (FN FN1)
   (COND
    (FN
      (SETQ DONE NIL)
      (TERPRI)
      (PRINSU (SETQ TREE (CAR (PROGST FN))))
      (TERPRI)
      (SETQ DONE (REVERSE DONE)) ))
   (COND
    (FN1
      (SETQ ALLCALLS NIL)
      (TERPRI)
      (PRIN1 FN1)
      (MAPC '(IS CALLED BY:) 'PRIN1)
      (TERPRI)
      (REVERSE (ALLCALLS FN1 TREE)) )
    (DONE)))

(DE REVERSE (L M)
  (COND
      ((NULL L) M)
      ((REVERSE (CDR L) (CONS (CAR L) M)))))

(DE PRINSU (X N)
    (OR N (SETQ N 1))
    (TTAB N)
    (COND
      ((ATOM X) (PRINT X))
      ((NULL (CDR X)) (PRINT (CAR X)))
      (T (PRIN1 (NEXTL X))
         (TTAB (SETQ N (PLUS N 7)))
         (MAPC X
               '(LAMBDA (X) (PRINSU X N) )) )) )

(DE PROGST (X D)
   (SETQ DONE (CONS X DONE))
   (LIST (CONS X (PRGSTR (GETD  X) ))))
```

Ce programme a été écrit par Daniel G. BØBRØW
voir : " A Program to document LISP Program Structure "
in LISP Bulletin, ACM SIGPLAN Notices, Vol 4 no9,
septembre 1969, pp 17-45.

```
(DE PRGSTR (X A)
   (COND
      ((ATOM X)
         (COND
             ((NOTFN X) NIL)
             ((MEMQ X D) NIL)
             ((MEMQ X DONE) (SETQ D (CONS X D)) (LIST X))
             ((SETQ D (CONS X D))
              (PROGST X)))))
      (T (SETQ A (CAR X))
         (COND
             ((MEMQ A '(LAMBDA PROG)) (PRGST1 (CDR X)))
             ((EQ A 'COND) (PRGST1 X))
             ((EQ A 'QUOTE) (PRGSTR (CADR X)))
             ((NCONC (PRGSTR A) (PRGST1 X)))))))


(DE NOTFN (FN)
   (ATOM (GETD FN)))


(DE PRGST1 (L A B)
   (WHILE (NEXTL L)
          (OR (ATOM (SETQ B (CAR L)))
              (SETQ A (NCONC A (PRGSTR B)))))
   A)


(DE ALLCALLS (FN TR A B)
   (SETQ A (CAR TR))
   (WHILE
      (NEXTL TR)
      (COND
        ((EQ (SETQ B (CAR TR)) FN)
          (SETQ ALLCALLS (CONS A ALLCALLS)))
      ((ATOM B))
      (T (AND (EQ (CAR B) FN) (SETQ ALLCALLS (CONS A ALLCALLS)))
         (ALLCALLS FN B) )))
   ALLCALLS)


(DE GETD (X) (OR (GET X EXPR) (GET X FEXPR)))
```

- EXEMPLE D'UTILISATION DE DIRECTORY -

```
(DIRECTORY 'PRINST)

PRINST  PRINSU  PRINSU
        PROGST  PRGSTR  NOTFN   GETD
                        PROGST
                        PRGST1  PRGSTR
                        PRGSTR
                GETD
        REVERS  REVERS
        ALLCAL  ALLCAL
```

Arbre des appels

```
PRINST IS CALLED BY:
NIL

PRINSU IS CALLED BY:
(PRINST PRINSU)

PROGST IS CALLED BY:
(PRINST PRGSTR)

PRGSTR IS CALLED BY:
(PROGST PRGST1 PRGSTR)

NOTFN IS CALLED BY:
(PRGSTR)

GETD IS CALLED BY:
(NOTFN PROGST)

PRGST1 IS CALLED BY:
(PRGSTR)

REVERS IS CALLED BY:
(PRINST REVERS)

ALLCAL IS CALLED BY:
(PRINST ALLCAL)
NIL


TREE ;;
 (PRINST (PRINSU PRINSU) (PROGST (PRGSTR (NOTFN (GETD)) PROGST (
PRGST1 PRGSTR) PRGSTR) GETD) (REVERS REVERS) (ALLCAL ALLCAL))
```

- TABLE DES MATIÈRES -